MICROSOFT®

TECHNICAL JOURNAL

Microsoft Confidential Editor, Ross Garmoe

April 11, 1984
Volume I

"This is the clearest piece of documentation I've ever written."

J. Joyce

Editor's Comments - Ross Garmoe

This issue of the journal is smaller than I had hoped it would be. Several people are planning to write articles describing their current work and tools environment but their commitments did not allow time to complete their articles for this issue. Future issues will contain a description of the Productivity Software tools environment by Bob Matthews and a discussion by Gabe Newell of the testing performed on the Macintosh software. I hope other groups will submit articles describing their work and tools environment.

Included with this issue is a manual page describing the program decode86 which is used to print 8086 object files in an intelligible format. This program is in /usr/tools on all of the 68K systems. This program, the C Merge 8086 Compiler and the Assembler described in a following article comprise the start of a complete cross-development package for all of the 68K systems. This package will also include the librarian, linker and upload/download programs developed by Systems Languages. Other utilities will be added to this package as needed and developed.

Also included in this issue is a chart showing the configuration of the Microsoft computer network. Because of the rate we are adding machines, this chart will soon be out of date if it is not already. However, most of the new machines will be added to the Ethernet so the basic configuration will remain the same.

Microsoft Project - Rob Glaser

SCHEDULING AND ORGANIZING PROJECTS WITH MICROSOFT PROJECT

"Efficiency isn't necessarily a capitalist plot."

Jerry Rubin
(Ex-Yippie)

We've just announced to the world a new project management productivity tool called Microsoft Project. In the days and weeks ahead you'll likely be hearing more about Project (which will be shipping by the end of May), but I wanted to take the opportunity here to describe some of the ways that Project might be useful internally.

Unlike many of the other tools described in this journal, Project doesn't help you write or debug code. Rather, Project is designed to help schedule the tasks you need to perform using the resources at your disposal. There are two key concepts central to the way Project works: One is a set of heuristics known as the **Work Breakdown Structure**, and the other is a scheduling technique called the **Critical Path Method**. Many readers are probably at least familiar with these two concepts, but I'll summarize below for those who aren't:

> **Work Breakdown Structure:** This is a world view that essentially looks at any project as a finite sequence of subsidiary tasks or activities.

Breaking down a big project into a set of discrete and interlocking activities is, at best, an art form. But it is an art form roughly analagous to the discipline of designing software modularly; in both cases a well-designed module (activity) has clearly defined inputs and outputs, performs a function that is well-defined, and should be as free of side effects as possible.

Critical Path Method: After representing a project's work breakdown structure, you need to schedule what gets done when (and by whom). CPM is the way that Project does this. CPM assumes that each activity has a known (or projected) duration, as well as a known (or posited) set of required predecessors (activities that must be completed before it can proceed) and successors (activities that cannot start before it has been completed). Given this information, CPM then tells you how soon you can and should start each of your activities and when, based on your input, the Project as a whole will be completed. Moreover CPM tells you which of your activities are "critical" -- any delay in these activities will delay the completion of the whole project.

It is fairly easy to see from the above how CPM scheduling and Microsoft Project can be applied to product development. For instance, consider the way that Steve Rowe, COBOL development manager, has been using Project. Steve manages a group of about eight programmers who are involved in several different COBOL-related projects such as COBOL 2.0, an ISAM package and COBOL Utilities. Steve breaks each of these projects down into tasks such as "design memory manager" and cooperatively with his development team determines how long each of these tasks (activities) ought to take. Based on these assessments, Steve builds a CPM schedule for each project which he uses for his own planning and scheduling. Moreover, Steve links each of these schedules into a master schedule (Project has an external linkage facility much like Multiplan's) so that he can get an overview of the whole department's progress.

Once the projects are underway, Steve uses Project as a tracking tool. If unforeseen delays come up, Steve can input the schedule changes to Project and see the overall implications of the delay. Steve has been using Project for about four months now.

People interested in using Project can get the latest copy of the disk off of my door (I'm currently in 2262) or can order it from the warehouse starting in mid-May (part #027-014). Project runs on IBM PCs or clones and requires 128K. It was developed jointly by Microsoft and MAS (a Seattle-based company). MAS wrote it in Lattice C on a development "environment" consisting of PC-XT's. Project will eventually be ported to other machines and environments, but as of this writing that porting strategy hasn't been finalized yet (but when we have a plan, you can bet that we'll schedule it with Project....)

Microsoft Assembler - Ross Garmoe

A test version of MASM, the Microsoft Assembler for the 8086, is available for testing on all of the 68K systems. The Assembler is accessible as /usr/tools/masm. This Assembler is a C implementation of the assembler that is available on the DEC 20 and MS-DOS. Any bug reports should be emailed to "rossg" along with the source of the program. MASM is executed in the following manner:

masm-options filename

The options are toggles with the following meanings and default settings:

flag	default	meaning of TRUE condition
a	FALSE	output segments in alphabetic order
c	FALSE	output cross reference data to filename.crf
d	FALSE	pass 1 listing to filename.lst
e	FALSE	use floating point emulation
h	TRUE	ouptu listing in hexadecimal
l	FALSE	listing (FALSE overrides d) to filename.lst
o	TRUE	output binary
x	FALSE	list errors to console

Exit codes have the following meanings:

code	meaning
0	no error
1	no file name specified
2	argument error
3	unable to open input file
4	unable to open listing file
5	unable to open object file
6	unable to open cross reference file
7	assembled in title too large
8	assembly errors

Notes from the Library - Natalie Yount

New Additions to the Library Collection (March 23 - April 4)

BOOKS

ARTIFICIAL INTELLIGENCE: AN MIT PERSPECTIVE. Patrick Winston and Richard Brown, editors. MIT Press, 1979.

A collection of MIT reports with sections on Expert problem solving, natural language understanding, understanding vision, computer design, symbol manipulation. 2 volumes.

ASSEMBLERS, COMPILERS AND PROGRAM TRANSLATIONS. Peter Calingaert. Computer Science Press, 1979.

"This book is designed primarily for use as a one-semester text by advanced undergraduate or beginning graduate students, but is suitable also for self-study by programmers and other computer professionals." Author's statement.

PROGRAMMER PRODUCTIVITY: MYTHS, METHODS AND MORPHOLOGY. A guide for managers, analysts and programmers. Lowell Arthur.

This guide offers productivity improvements of 100 - 1000% and includes numerous quotes and anecdotes (for example, on electronic mail, "'Out of sight, out of mind,' when translated into Russian by computer then back again into English (becomes) 'invisible maniac.'" Look here for good quotes if not productive reading activity.

THE ICON PROGRAMMING LANGUAGE. R. Griswold. Prentice-Hall, 1983.

"Continuing the tradition of SNOBOL4, Icon has modern control structures, and its facilities for string analysis and synthesis are more extensive than those of SNOBOL4. While string analysis and synthesis, together with goal-directed evaluation, are just a part of the pattern matching mechanism in SNOBOL4, these features are an integral part of Icon, making possible a wide variety of programming techniques." Author's statement.

COMPUTER ARCHITECTURE AND ORGANIZATION. John Hayes. McGraw-Hill, 1978.

MEASUREMENT AND TUNING OF COMPUTER SYSTEMS. D. Ferrari. Prentice Hall, 1983.

PROGRAMMING LANGUAGES: DESIGN AND IMPLEMENTATION. 2nd edition. Terrence Pratt. Prentice-Hall, 1984.

SCREEN DESIGN STRATEGIES FOR COMPUTER ASSISTED INSTRUCTION. J. Heines. Digital Press, 1984.

Introduces the major concepts in designing video displays for CAI. Sections include: standard screen components, visual symbols, menus, text display, screen design tools, CAI style.

WHACK ON THE SIDE OF THE HEAD: HOW TO UNLOCK YOUR MIND FOR INNOVATION. Roger von Oech. Warner Books, 1983.

"A book about the ten mental locks that prevent you from being more innovative - and what you can do to open them."

NEW TECHNICAL REPORTS

BROWN UNIVERSITY

Marc Brown and Robert Sedgewick. A SYSTEM FOR ALGORITHM ANIMATION. CS-84-01 January 1984.

Marc Brown and Robert Sedgewick. TECHNIQUES FOR ALGORITHM ANIMATION. CS-84-02 January 1984.

Joseph Pato, Steven Reiss and Marc Brown. THE BROWN WORKSTATION ENVIRONMENT. CS-84-03 January 1984.

Steve Reiss. GRAPHICAL PROGRAM DEVELOPMENT WITH PECAN DEVELOPMENT SYSTEMS. CS-84-04 February 1984.

Scott Smolka. ANALYSIS OF COMMUNICATING FINITE STATE PROCESSES. May 1984.

STANFORD REPORTS

Donald Alpert. PERFORMANCE TRADEOFFS FOR MICROPROCESSOR CACHE MEMORIES. December 1983

Frederick Chow. A PORTABLE MACHINE-INDEPENDENT GLOBAL OPTIMIZER - DESIGN AND MEASUREMENTS. December 1983.

Michael Fine. PERFORMANCE OF UNIDIRECTIONAL BROADCAST LOCAL AREA NETWORKS; EXPRESS NET AND FASNET. December 1983.

Jeffrey Ullman. SOME THOUGHTS ABOUT SUPERCOMPUTER ORGANIZATION. October 1983.

If you would like to order any of these, contact the Library.

LIBRARY DATABASE

For several months we have been entering data into the Library database. The database will contain all materials held by or acquired through the Microsoft Library including books, manuals, technical reports, standards, articles. We are using the UNIFY database. The database is intended to provide flexible access to the information resources held by Microsoft. Currently there is an access terminal in the Library which replaces the traditional card catalog. This type of access, although better than no access, does not seem to me to be flexible enough to meet the information needs at Microsoft. I would like to solicit your help. Please think about the following and send mail or come in with ideas which might help me provide better information access to the MS community.

Do people really need to be able to search the system themselves or is it enough to send mail to the Library and have us send the information? (Remember we have greater access than just to information that Microsoft holds.)

Is it feasible or even necessary for people to be able to access the database from their own work areas or will the "electronic" card catalog in the library suffice?

What needs to be done to make access to the library database from remote locations possible? Is it worth it?

Column C - Ralph Ryan and Hans Spiller

WRITING EFFICIENT C CODE

This column is about writing efficient C code. We will deal with ways to make the compiled code both smaller and faster. The techniques discussed here should be efficient in most C compilers, and are known to be so in the C Merge C compilers. To belabor the obvious, it is most important to get the code correct. The techniques presented here represent a set of good habits for efficient coding.

REGISTER VARIABLES

Probably the single greatest efficiency is register variables. When used correctly, they always produce smaller faster code. There is a cost involved with a register variable: that register must be saved and restored at procedure entry and exit. A rule of thumb across machines is that two references to a variable in a procedure is the breakeven point. All local variables (including parameters) may be declared as register variables. Only "int" sized variables and "int" sized pointer variables are guaranteed to be used across machines.

The number of register variables available is machine dependent: the 8086 has 2, the PDP-11 has 3, and the 68000 has 9 (5 data and 4 address).

The compiler guarantees to assign register variables in exactly the same order as they are encountered in the source. Thus register variables should be declared in their order of importance. One anomaly arises, suppose that in portable code a parameter is the third most important register variable. On the 8086, the register declaration would be honored over subsequent declarations. There are two tricks for getting around this:

```
foo(a)
            int A:
            (
            register int b:
            register int c;
            register int d = a
            )
------------------
#define REG1
#define REG2   register
#define REG3   register

foo(a)
            REG3 a;
            (
            REG1 b;
            REG2 c;
            )
```

A machine dependent include file would be used to define the REGx macros.

TEMPORARY VARIABLES

In the absence of common subexpression elimination (CSE is 6-9 months away in the C Merge compilers), often used subexpressions can be put into a temporary variable. If the temporary is a register variable the gains are even greater. Example:

```
p->a->b->c = 1;
p->a->b->d = 1;
```

can become

```
q = p->a->b;
q->c = 1;
q->d = 1;
```

The threshold of when it is a win to do this depends upon the relative complexity of the various expressions, and the target machine. In general two references will be pretty close, and three will always win.

A FEW TRICKS

Assigning operators (+=, -=, etc.) are almost always an improvement. C Merge attempts to rewrite expressions as assignment operators if it can, but beyond some level of complexity it cannot tell about side effects.

Divide and remainder can often be replaced by shift and mask. If the operands are signed values, the compiler cannot do this optimization automatically, because it is not valid for negative numbers that are not a power of two. Either use 'unsigned' variables so if you can, or do the optimization yourself.

In line with this, multiplying by a power of 2 can be replaced by a shift. Remember that there is a multiply in all array references. So if you make the elements of arrays that are referenced heavily have a size that is a power of 2, you will get better code. You will have to make the decision whether its worth wasting the space to pad the element size to a power of 2 if necessary.

Given a choice of using a 'switch' or a series of 'if', 'else if' ..., 'else', where a value or expression is compared against a number of constants, use the 'switch'.

```
if (*p == 'a' || *p == 'b' || *p == 'c')
                .....
```

evaluates the pointer indirection each time (of course CSE will improve this).

```
switch(*p)
        (
        case 'a' :
        case 'b' :
        case 'c' :
                ...
        )
```

only evaluates the indirection once. If the set of comparisons is "dense" (number of cases/range of cases > 50 percent then it is even more efficient: the generated code indexes through a branch table. If there are two or more constants involved, use a switch.

WHAT TO AVOID

Unless precise real calculations are required, floating point arithmetic should be avoided. It is slow and large on almost all microcomputers. If floating point calculations are required, stay away from "float" declarations — since C requires that all calculations must be done as "double", it avoids unnecessary and expensive type conversions.

BUFFERED I/O

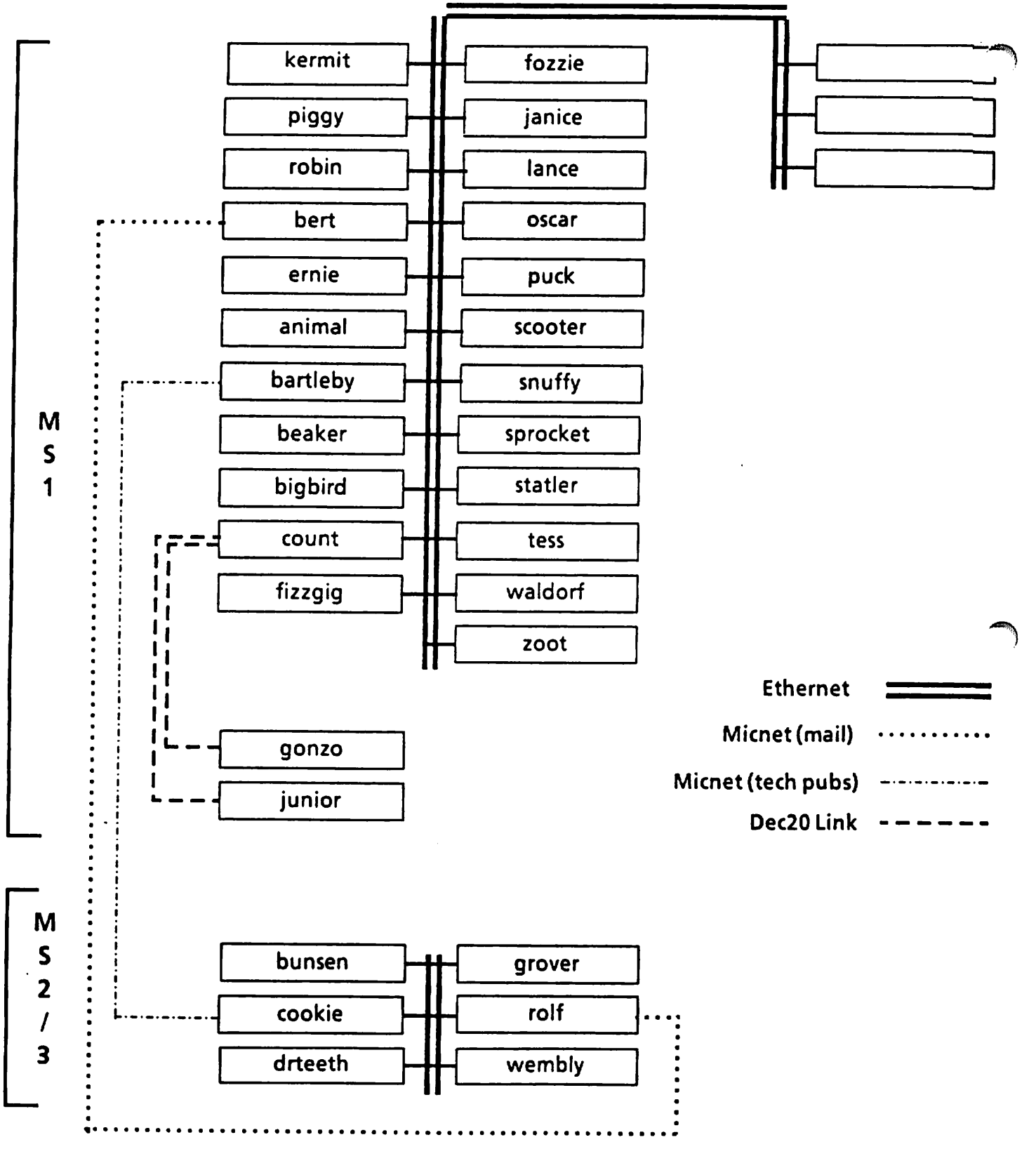
The standard C library (stdio) provides a buffered I/O package. Because much I/O to the terminal is interactive, stdio takes the philosophy that writing to the terminal should go unbuffered. The "setbuf" and "fflush" calls cause the I/O to always be buffered, and flush only when requested. This is almost always a significant performance improvement.

PERFORMANCE

What we have presented here is only part of writing efficient (and readable) code. In the first paragraph a mention was made to measuring the performance of a program. On both XENIX and MS-DOS there are profiling tools available that will allow you to see where the inefficiencies are in a program. In addition, there is a file with some newly developed profiling tools being released to /usr/tools in all the in-house XENIX systems. This set of subroutines allows you to instrument a program and start, suspend, resume, and print the values of 5 separate profiling clocks. The next Column C will give more details on using this profiling package.

The source to these profiling routines are in /usr/tools/src/profile and are called profile.c and profile.h. The relocatable object is /usr/tools/profile.o.

* * * * *


kermit
fozzie
piggy
janice
robin
lance
bert
oscar
ernie
puck
animal
scooter
bartleby
snuffy
beaker
sprocket
bigbird
statler
count
tess
fizzgig
waldorf
zoot
gonzo
junior
MS1
bunsen
grover
cookie
rolf
drteeth
wembly
MS2/3
Ethernet
Micnet (mail)
Micnet (tech pubs)
Dec20 Link